Pour tous les exercices : lorsque l'on vous demande d'analyser la complexité, on se réfère à la complexité dans les cas extrêmes (« worst case time complexity »).

Exercice 1
Analysez la complexité pour les deux codes suivants. Est-ce la même pour fun() et fun2() ? Pourquoi ?

```
def fun(n):
    for i in range(n):
        for j in range(n):
            print (n)

def fun2(n):
    for i in range(n):
        print (n)
    for j in range(n):
        print (n)
```

Exercice 2
Quel est le niveau de complexité de ce code? Comment obtenez-vous cette complexité ?

```
def f(n):
    if n <= 1:
        return 1
    else:
        return f(n-1) + f(n-1)
```

Exercice 3 :
Un programme Python prend un entier (« integer ») 2^N, N > 0 en entrée (e.g., 16) et successivement divise le nombre par deux jusqu'à obtenir 1 (voir ci-dessous). Analyzer la complexité de ce code.

```
def fun(k):
    if k == 1:
        print ('Done')
    else:
        k = k/2
        fun(k)
```

Exercice 4 :
On vous donne une liste d'entiers trié par ordre ascendant (« sorted integers ») en entrée. Ecrivez un code qui retourne la valeur maximum de cette liste. Aide: la fonction len() a une complexité de O(1).
Exemple avec [2,3,5,9] : l'élément maximum est 9.

Exercice 5 :
En cours, vous avez vu que les listes tel [1,2]. Il est également possible de créer une liste dans une liste (liste imbriquée ou « nested list ») tel nested_list=[[1,2],[3,4]]. Si vous souhaitez accéder à l'élément 3 de la variable nested_list vous pouvez le faire ainsi nested_list[1][0] (1 pointant sur la deuxième liste [3,4] et 0 pointant sur le premier élément de cette dernière).

Ecrivez un programme python qui retourne l'élément maximum et minimum d'une liste imbriquée.
Aide : Une boucle imbriquée (une boucle dans une boucle) pourrait être utile pour accéder à chaque entier de la liste imbriquée.

Exercice 6 :
On vous donne un entier, n, et une liste d'entiers triés, x. Ecrivez un programme Python qui cherche dans la liste y le plus petit élément qui est supérieur à x.
Exemple avec n=700 et y=[2,5,10,44,130,542,940,2319,40194,229292] : le plus petit élément de la liste supérieur à 700 est 940.
Quelle est la complexité de votre programme? Peut-elle être réduite ?

Exercice 7 :
On vous donne un nombre entier, n, et une liste triée, y. Ecrivez un programme qui insère la valeur n dans la liste y de sorte à ce que la liste y reste triée.
Exemple avec n=5 et y=[2,4,6] : résultat=[2,4,5,6]

Exercice 8 :
On vous donne une liste de caractères (« strings ») qui représentent des entiers (« integers »). Par exemple : ['1', '9' ,'4', '15'] (notez la différence avec une liste d'entiers [1, 9, 4, 15]). Ecrivez un programme qui retourne la valeur maximum et minimum de la liste.
Exemple : ['1', '9' ,'4', '15'] doit retourner 1 et 15.
Aide : comparaison de caractères '9' > '15' alors que comparaison d'entiers 9 < 15.

Exercice 9 :
Implémentez l'algorithme « Bubble sort » en vous aidant du descriptif ci-dessous (source : https://openclassrooms.com/fr/courses/1160591-le-tri-a-bulles)

> *"Le principe de l'algorithme du tri à bulles est très simple à assimiler. Il est, et de loin, l'un des algorithmes de tri les plus simples qui soient.*
>
> *Pour vous expliquer le principe, je vais d'abord vous donner une courte explication écrite puis nous allons concrètement trier une liste de nombres.*
>
> *Le principe du tri à bulles est de comparer deux valeurs adjacentes et d'inverser leur position si elles sont mal placées. Alors, qu'entend-t-on par "mal placé" ? C'est très simple et surtout, c'est logique : si un premier nombre x est plus grand qu'un deuxième nombre y et que l'on souhaite trier l'ensemble par ordre croissant, alors x et y sont mal placés et il faut les inverser. Si, au contraire, x est plus petit que y, alors on ne fait rien et l'on compare y à z, l'élément suivant. C'est donc itératif. Et on parcourt ainsi la liste jusqu'à ce qu'on ait réalisé n-1 passages (n représentant le nombre de valeurs à trier) ou jusqu'à ce qu'il n'y ait plus rien à inverser lors du dernier passage.*
>
> *Avec de la logique, on s'aperçoit qu'au premier passage, on place le plus grand élément de la liste au bout du tableau, au bon emplacement. Pour le passage suivant, nous ne sommes donc plus obligés de faire une comparaison avec le dernière élément ; et c'est bien plus avantageux ainsi. Donc à chaque passage, le nombre de valeurs à comparer diminue de 1."*
>
> *Source: https://openclassrooms.com/fr/courses/1160591-le-tri-a-bulles*

Exercice 10 :
Implémentez l'algorithme de « selection sort » en vous aidant du descriptif ci-dessous :

- Imaginez deux listes: La liste A contient des éléments entiers en désordre et la liste B est une liste vide

- Scannez la liste A à la recherche du plus petit élément. Supprimez cet élément de la liste A et insérer le à la fin de la liste B.

- Répétez l'étape ci-dessus jusqu'à ce que A soit vide.